数据结构

第一次上机作业实验文档

学号2017211123

班级 2017211301

序号 17

姓名 褚逸豪

## **问题描述**

输入表达式字符串，以“=”表示结束， 计算并输出表达式值。操作数可以是整数或实数，操作符有 “+”、“-”、“\*”、“/”、“^”（乘方）和 “sin( )”（正弦）、“cos( )”（余弦）、“log( )”（对数）、“ln( )”（自然对数）等函数。

## 算法思路

使用迪杰斯特拉双栈表达式解析算法，使用两个栈：符号栈和数字栈，数字栈储存运算符栈中运算符所对应的操作数，符号栈中存储表达式运算符号。在解析表达式的时候，我们确保之前优先级更高的运算单元已经被计算完毕。计算一个单元，我们取出一个运算符，取出相应数量的操作数，计算出结果，重新压入数字栈，即完成了一个单元的计算。当整个表达式读取完毕，我们只需循环计算单元，直到操作数栈中不在有内容为止。

源串的处理，源串进行关键字替换后变成准标准串，准标准串中使用单个英文字符来代表函数。由于Log是仅有的双参数函数，因此，用”,”来识别log函数。

准标准串经过去空格之后得到标准串，对标准串使用迪杰斯特拉双栈表达式解析算法，可得表达式的值。

## 算法描述

1. 读取源串
2. 将源串进行关键字替换
3. 去除空格
4. 开始读入
   1. 判断接下来是实数还是运算符，并交由不同部分处理
   2. 若是实数直接压栈，否则判符号栈栈顶元素与当前运算符的优先级关系，若当前运算符具有更高的优先级，则压栈，否则进行运算弹栈，遇到左括号无条件压栈，右括号循环运算弹栈，直到栈顶遇到左括号。
5. 循环运算弹栈，直到符号栈为空，数字栈中的栈顶元素即为表达式的值

## 源程序及驱动程序

编译命令：g++ <文件名.cpp> -std=c++11

（需开启c++11编译开关）

#include <cstring>

#include <iostream>

#include <cctype>

#include <cmath>

#include <regex>

#include <string>

namespace dij\_exp\_eval {

const int MXN = 100007;

double base[33] = {1}; // 小数数位基，在读入小数时会用到

std::map<std::string, std::string> lst = {

{"cos", " C "},

{"sin", " S "},

{"tan", " T "},

{"sqrt", " R "},

{"ln", " L "},

{"floor", " F "},

{"log", " "},

{"exp", " E "},

{",", " l "}

}; // 建立单参数函数与字符的关联，将函数转化为运算符处理

template <typename T>

struct stk {

T a[MXN];

int t;

std::string nme;

stk(std::string nme): nme(nme) { memset(a, 0, sizeof(a)); t = 0; }

void push(T x) {

#ifdef DEBUG // 调试开关

std::cout << nme << " Instack:\t" << x << std::endl;

#endif

a[t++] = x;

}

T top() { return a[t - 1]; }

T pop() {

#ifdef DEBUG // 调试开关

std::cout << nme << " Outstack:\t" << a[t - 1] << std::endl;

#endif

return a[--t];

}

int size() { return t; }

bool empty() { return t <= 0; }

}; // 固定大小的模板栈类

stk<double> num("NUM");

stk<char> ops("OPS");

int pri[256]; // 运算符优先级表

double calc() { // 消除符号栈中的一个运算符号，并根据运算符消去并添加相应元素到数栈

char op = ops.top();

double b = num.top();

ops.pop();

num.pop();

if (op == 'L') { // ln

return log(b);

} else if (op == 'S') { // sin

return sin(b);

} else if (op == 'C') { // cos

return cos(b);

} else if (op == 'T') { // tan

return tan(b);

} else if (op == 'R') { // sqrt

return sqrt(b);

} else if (op == 'F') { // floor

return int(b);

} else if (op == 'E') { // exp

return exp(b);

}

double tmp = num.top();

num.pop();

if (op == '+') {

return tmp + b;

} else if (op == '-') {

return tmp - b;

} else if (op == '\*') {

return tmp \* b;

} else if (op == '/') {

return tmp / b;

} else if (op == '^') {

return pow(tmp, b);

} else if (op == 'l') {

return log(b) / log(tmp);

}

}

char s[MXN];

double analyze(const char\* sss) { // 准标准表达式解析

memset(s, 0, sizeof(s));

while (!num.empty()) num.pop();

while (!ops.empty()) ops.pop();

int len = 0, tmp = 0, i, j, k, f;

double ftmp = 0;

for (i = 0; sss[i] && sss[i] != '='; ++i) if (pri[sss[i]] || isdigit(sss[i]) || sss[i] == '.') s[len++] = sss[i]; // 消去空格

// printf("%s\n", s); // 打印标准化表达式

for (i = 0, f = 1; i < len; ++i, f = 1, tmp = 0, ftmp = 0) {

if (isdigit(s[i])) { // 处理数字

tmp = s[i] - '0';

HELL:;

for (j = i + 1; j < len && isdigit(s[j]); ++j)

tmp = tmp \* 10 + s[j] - '0';

if (s[j] == '.') { // 小数处理

for (j++, k = 1; isdigit(s[j]) && j < len; ++j, ++k) {

ftmp += base[k] \* (s[j] - '0');

}

}

i = j - 1;

ftmp += tmp;

num.push(ftmp \* f);

} else if (pri[s[i]]) {

if (s[i] == ')') { // 立即处理当前括号对

while (ops.top() != '(')

num.push(calc());

ops.pop();

} else if (s[i] == '-' && (i == 0 || i && !isdigit(s[i - 1]) && s[i - 1] != ')')) { // 捕获到负数，交由数字处理部分进行处理

f = -f;

goto HELL;

} else if (ops.empty() || s[i] == '(' || pri[ops.top()] < pri[s[i]] || (pri[ops.top()] >= 5 && pri[ops.top()] == pri[s[i]])) { // 压运算符栈

ops.push(s[i]);

} else {

while (!ops.empty() && pri[ops.top()] >= pri[s[i]]) // 处理栈顶元素直到满足压栈要求

num.push(calc());

ops.push(s[i]);

}

}

}

while (!ops.empty()) // 解算最终结果

num.push(calc());

return num.top();

}

bool inittag = false; // 第一次调用时将进行初始化

double solve(char rs[]) {

if (!inittag) {

for (int i = 1; i < 33; ++i) base[i] = base[i - 1] / 10; // 初始化小数数位基

pri['('] = 1; // 初始化优先级表

pri['l'] = 2;

pri['+'] = 3;

pri['-'] = 3;

pri['\*'] = 4;

pri['/'] = 4;

pri['^'] = 5;

pri['C'] = 6;

pri['S'] = 6;

pri['L'] = 6;

pri['R'] = 6;

pri['E'] = 6;

pri['F'] = 6;

pri['T'] = 6;

pri[')'] = 20;

}

std::string s(rs);

for (auto const &i : lst)

s = std::regex\_replace(s, std::regex(i.first, std::regex::icase), i.second); // 将函数字符串替换成字符

return analyze(s.c\_str()); // 解析表达式

}

double solve(std::string s) {

if (!inittag) {

for (int i = 1; i < 33; ++i) base[i] = base[i - 1] / 10; // 初始化小数数位基

pri['('] = 1; // 初始化优先级表

pri['l'] = 2;

pri['+'] = 3;

pri['-'] = 3;

pri['\*'] = 4;

pri['/'] = 4;

pri['^'] = 5;

pri['C'] = 6;

pri['S'] = 6;

pri['L'] = 6;

pri['R'] = 6;

pri['E'] = 6;

pri['F'] = 6;

pri['T'] = 6;

pri[')'] = 20;

}

for (auto const &i : lst)

s = std::regex\_replace(s, std::regex(i.first, std::regex::icase), i.second); // 将函数字符串替换成字符

return analyze(s.c\_str()); // 解析表达式

}

}

int main() {

std::string s;

while (true) {

std::getline(std::cin, s);

if (!std::cin) break;

std::cout << dij\_exp\_eval::solve(s) << std::endl;

}

}

## 测试数据

### Input

1+2+3=

log(10,100\*45)^3+2/3-cos(10\*3.1415926)+sqrt(floor(4.5))=

log(2\*2+1,exp(ln(5)))=

(exp(100)-exp(99))/2=

### Output

6

50.4223

1

8.49607e+42

### Online Test

<https://ideone.com/Emhh4J>

（可以fork之后修改输入，直接在线运行出结果）

## 结果分析

结果与Casio 82EPS计算器所计算出的结果在输出的精度下没有误差

## 结论

本程序在输入形式正确的情况下可以给出算式的正确计算结果

## 心得体会

我在写这个表达式解析的程序的时候使用了一些trick，我将函数名替换成了单个字符，这样就可以把他们当作普通的操作符进行处理了。由于函数中，只有log函数是具有两个变量的函数，因此我直接将逗号视作log的运算符，并且具有比加减还要低的优先级。